Угнать за 5 миллисекунд:

как мы делали транспорт для торговых ядер Московской биржи

Николай Карлов и Олег Уткин





Кто мы?

Отдел архитектуры систем хранения данных Mail.Ru

Делаем пилоты и прототипы систем с нетривиальными требованиями

Олег Уткин — разработчик

Николай Карлов — главный архитектор



О чем будем говорить?

В прошлом году Московская биржа обратилась к нам с задачей разделения торговых ядер

Мы сделали совместный пилот системы доставки торговых данных

… и он оказался успешным

Расскажем об этом опыте, архитектуре и выученных уроках



Что такое “разделить ядра”?

Это дать возможность вынесения торгов на отдельные торговые ядра:

a. По инструменту

b. По ценной бумаге

c. По какой-либо еще логике

Пример: сделать листинг очень крупной компании на отдельном ядре



Почему это важно?

- Горизонтальное масштабирование

  (сейчас Биржа умеет горизонтально масштабировать только чтение)

- Минимизация рисков при листинге



Какие главные требования?

- Клиенты не должны заметить разделения
- Все должно очень быстро и надежно работать
- А еще должно масштабироваться как на чтение, так и на запись



Как работает биржа

4 3 2 Заявки

1

МАСТЕР

торговое ядро

Заявка (ордер):

“Хочу купить $10000 за рубли”

Попадает в биржевой стакан



Как работает биржа

4 3 2

Заявки

1

мастер

торговое ядро

Встречается с другой заявкой

В результате формируется сделка

заявка частично или полностью удовлетворяется

... или через некоторое время снимается



Как работает биржа

4 3 2

Заявки

1 1 1

МАСТЕР

торговое ядро

торговое ядро

торговое ядро

мастер

реплика

реплика



Как работает биржа

4 3 2

заявки

1 1 1

МАСТЕР

торговое ядро

торговое ядро

торговое ядро

Сервера доступа

Брокеры

Трейдеры

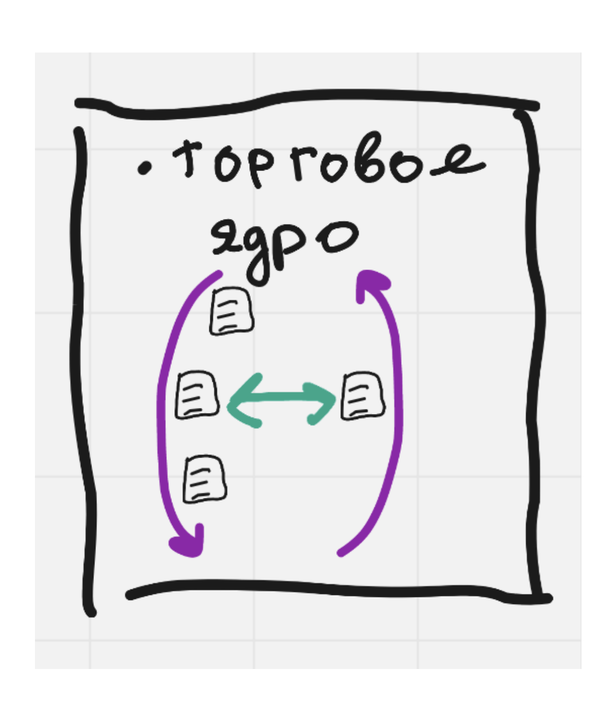
get

ок

ок



В ядре есть два глобальных счетчика



1. Rec (record number)

Увеличивается при создании заявки

2. Seq (sequence number)

Увеличивается при обновлении любой заявки



В любой заявке есть эти два поля

Структура заявки (~600B)

FirmID	идентификатор фирмы
Rec	**номер заявки**
Seq	**сквозной номер апдейтов**
OrderID	ИД заявки
...	прочие метаданные

Обновления происходят по OrderID

Запросы без фильтров по (rec, seq)

Запросы с фильтрами по (FirmID, rec, seq)



А как клиенты читают данные?

Дай мне самые свежие заявки без пропусков

(Rec, seq)

торговое ядро

Свежие:

- Новые
- Недавно изменившиеся



А как клиенты читают данные?

Дай мне самые свежие заявки без пропусков

Я знаю о существовании заявок с номером **rec**

(Rec, seq)

торговое ядро

И я знаю, что в бирже был апдейт **seq**

Свежие:

- Новые
- Недавно изменившиеся



А как клиенты читают данные?

Дай мне самые свежие заявки без пропусков

(Rec, seq)

торговое ядро

Клиент получает список заявок

Пара (rec, seq) из последней заявки используется для следующего запроса

(rec, seq) – continuation token

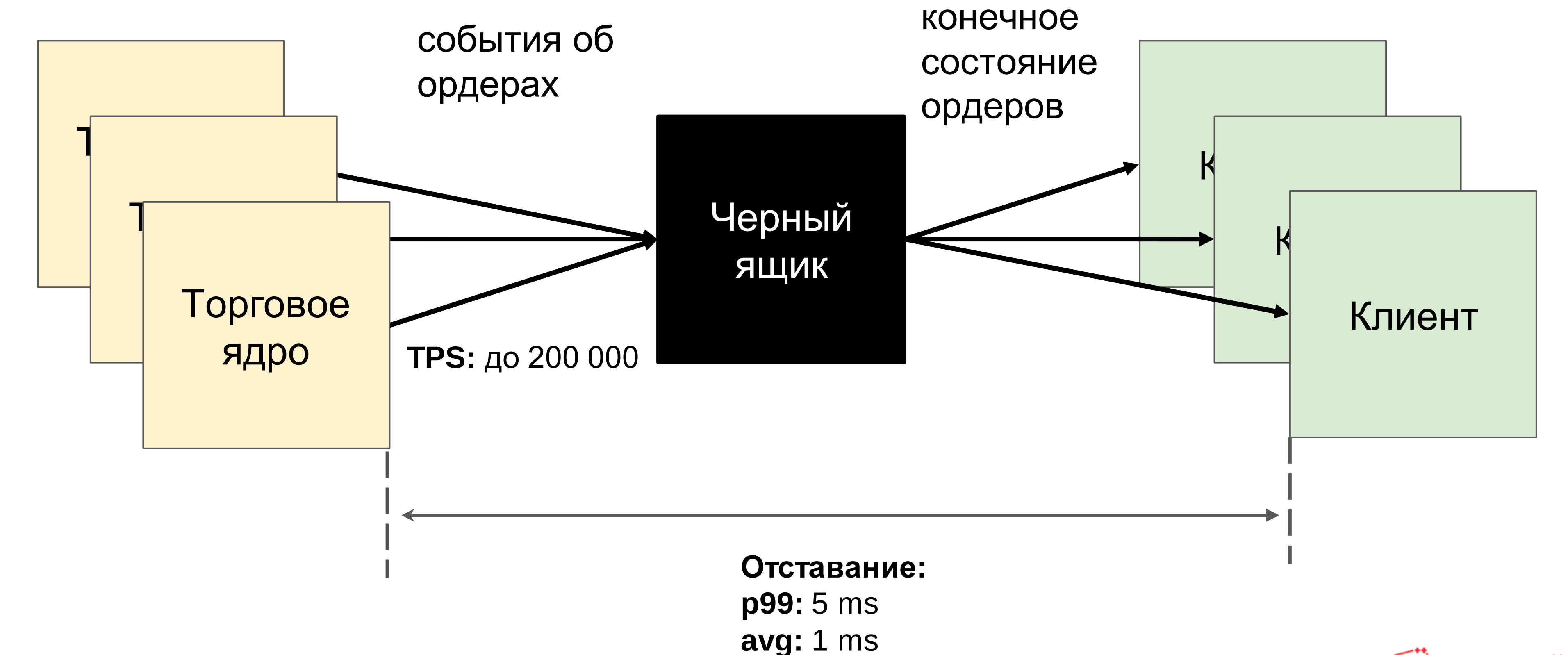


Что мы поняли?

- Клиенты получают данные поллингом
- Необходимо отдавать самые свежие (созданные или изменившиеся) заявки без пропусков
- Есть запросы на чтение и запись по вторичным ключам



А что с насчет производительности?



Почему не совсем очередь

- При апдейте ядро отдает только изменившиеся поля

  А клиенты хотят сущности целиком

- Если клиент отстал, то он не хочет получать все состояния сущностей

  Клиент хочет только последнее состояние



Сводим воедино требования к архитектуре

- Удельная производительность > 200 000 TPS на 1 узел
- Отставание от источника p99 < 5мс, а среднее — 1ms
- Клиент не должен знать о разделении
- Должен гарантироваться порядок данных
- Гарантия получения самых свежих данных без пропусков
- Одинаково хорошо отдавать как горячие (в пределах минуты), так и холодные (от минуты до дня)



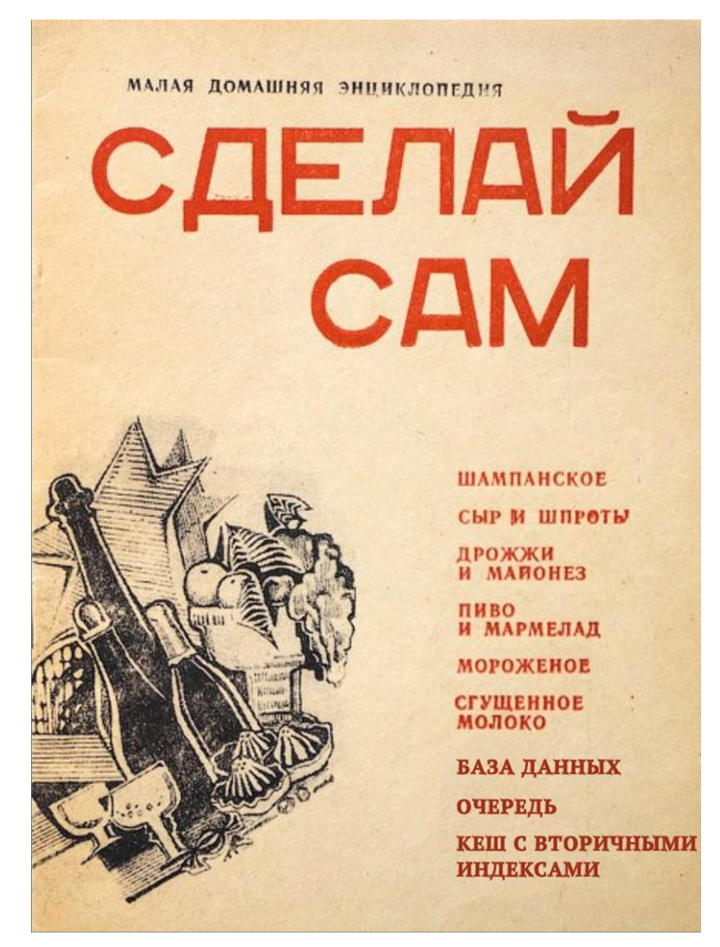
И тут мы понимаем,

что это должна быть одновременно:

- **база данных**,
- **очередь** с индексами
- **кэш с вторичными индексами**

А еще:

- с гарантиями сохранности данных
- чрезвычайно быстрая
- с гарантиями latency
- распределенная





Появилась гипотеза, что Tarantool может подойти для этой задачи

- Тоже однопоточный и in-memory, как и ядро биржи =)
- Написан на Си
- Имеет асинхронный сетевой протокол с мультиплексированием
- Фактически это сервер приложений с кооперативной многозадачностью
- Есть примитивы хранения и доступа к данным
    - Таблицы
    - Вторичные и составные ключи
- Есть хранение данных на диске и восстановление
- Есть репликация и шардинг
- И главное: это фреймворк для создания СУБД



Счетчик целесообразности выбора технологии







Технология подходит

0 : 0

Технология не подходит



Структура данных

OrderID	первичный ключ
Rec	**номер заявки**
Seq	**номер изменения**
Order time	время создания заявки
Update time	время обновления заявки
Binary blob	прочие метаданные

} ключ пагинации

insert → Rec++

update → Seq++



Сводим воедино требования к архитектуре

- Удельная производительность > 200 000 TPS на 1 узел
- Отставание от источника p99 < 5мс, а среднее — 1ms
- Клиент не должен знать о разделении
- Должен гарантироваться порядок данных
- **Гарантия получения самых свежих данных без пропусков**
- **Одинаково хорошо отдавать как горячие (в пределах минуты), так и холодные (от минуты до дня)**



Составной индекс

field1	field2
1	1
	2
	3
2	1
	2
	3
3	1
	2



Составной индекс

(2, 3)

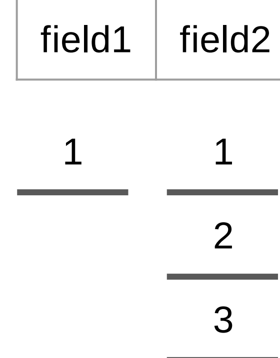
field1	field2
1	1
	2
	3
2	1
	2
	3
3	1
	2

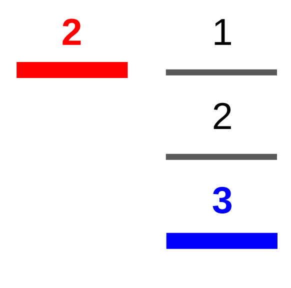


Составной индекс

(2, 3)

Поиск первой записи – O(logN)





B+tree

(древесный индекс с прошитыми листами)



Составной индекс

(2, 3)

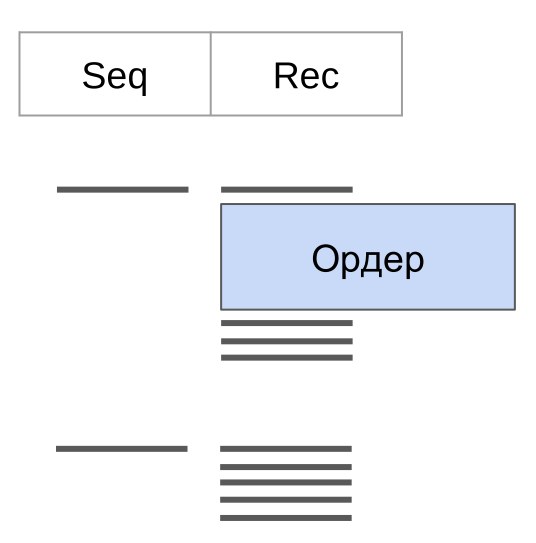
field1	field2
1	1
	2
	3
2	1
	2
	3
3	1
	2

Итерирование по связному списку O(1)

Очень эффективно



Составной индекс





Составной Индекс (update)

Seq	Rec

Старый ордер

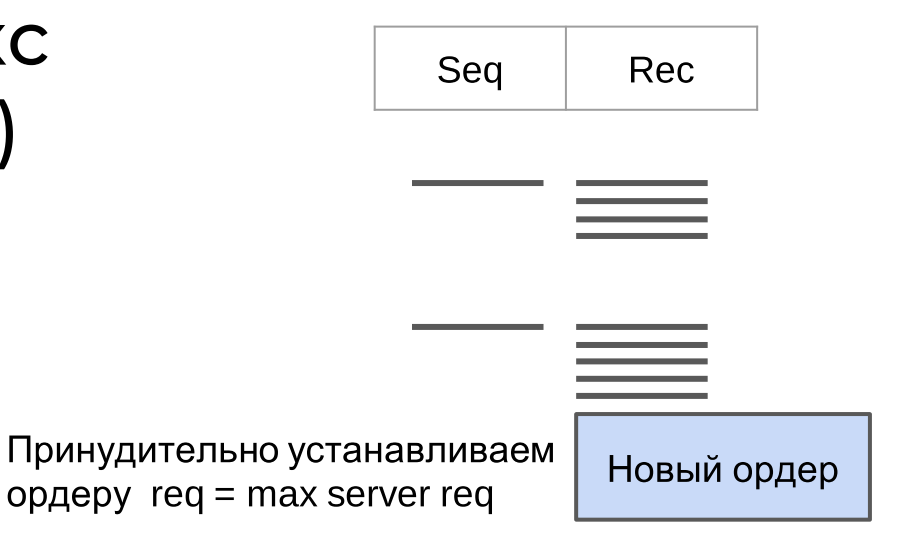
Update

Обновленный ордер

Свежие данные всегда в конце индекса



Составной Индекс (insert)



Свежие данные всегда в конце индекса!



Сова пока побеждает







1 : 0



Верхнеуровневая архитектура

Эмулятор торгового ядра

Эмулятор торгового ядра

Эмулятор
торгового
ядра

события об
ордерах

Хранилище
на базе Tarantool

конечное
состояние
ордеров

Эмулятор клиента

Эмулятор клиента

Эмулятор
клиента



Верхнеуровневая архитектура

Эмулятор торгового ядра

Эмулятор торгового ядра

Эмулятор торгового ядра

события об ордерах

Хранилище на базе Tarantool

конечное состояние ордеров

Эмулятор клиента

Эмулятор клиента

Эмулятор клиента



Верхнеуровневая архитектура

Эмулятор торгового ядра

Эмулятор торгового ядра

Эмулятор торгового ядра

события об ордерах

Хранилище на базе Tarantool

конечное состояние ордеров

Эмулятор клиента

Эмулятор клиента

Эмулятор клиента



Первая версия реализации

Tarantool App

Торговое ядро

push

события об ордерах

ApplyEvent

insert or update order

insert/update order

GetOrders

get orders after key

select orders

Space (table)

get

Клиент

конечное состояние ордеров



Максимальный поток транзакций

TPS	CPU, %		Комментарий
	основной поток	сетевой поток	
340k	100	15	При потоке без ограничений все начинается с 390k, а потом идет деградация до 340k
250k	72	29	
170k	49	23	
85k	24	12	

1 инстанс Tarantool способен выдержать поток в 340k TPS.



Сова пока побеждает







2 : 0



Рост отставания клиентов







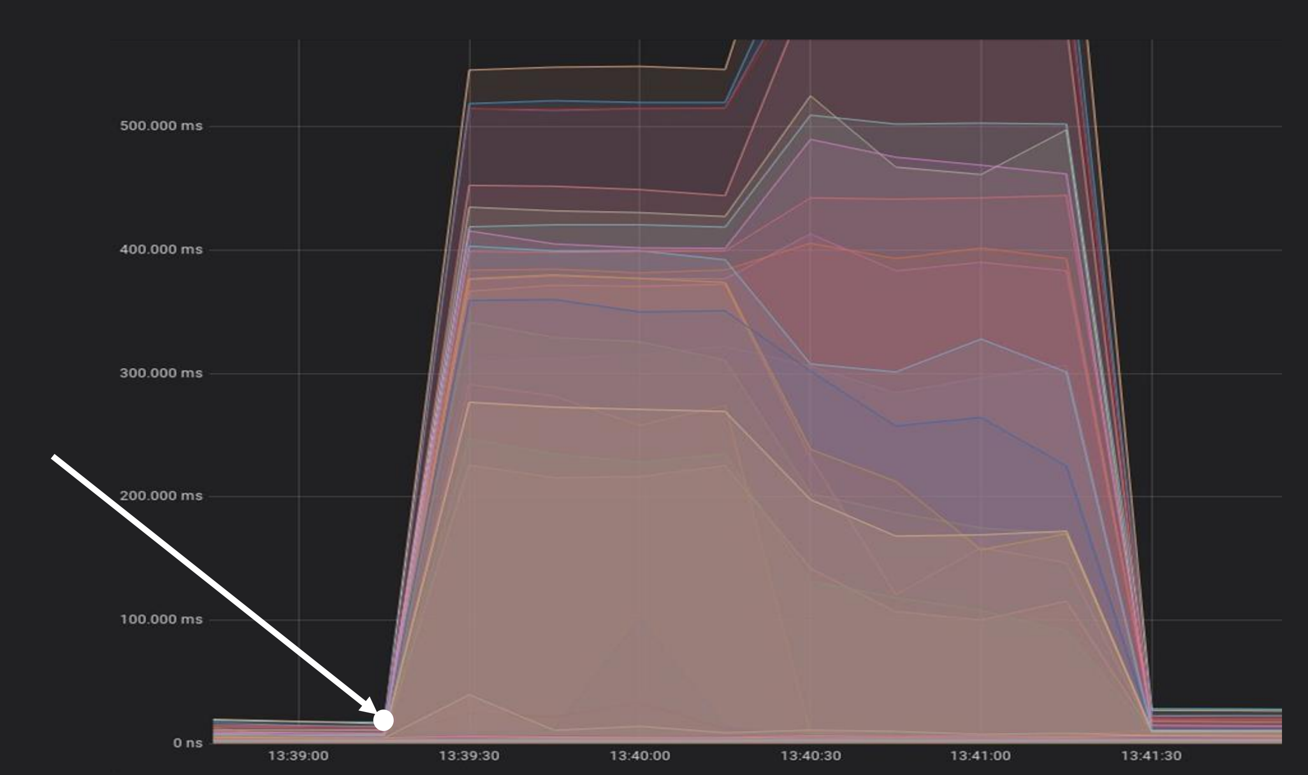


2 : 1

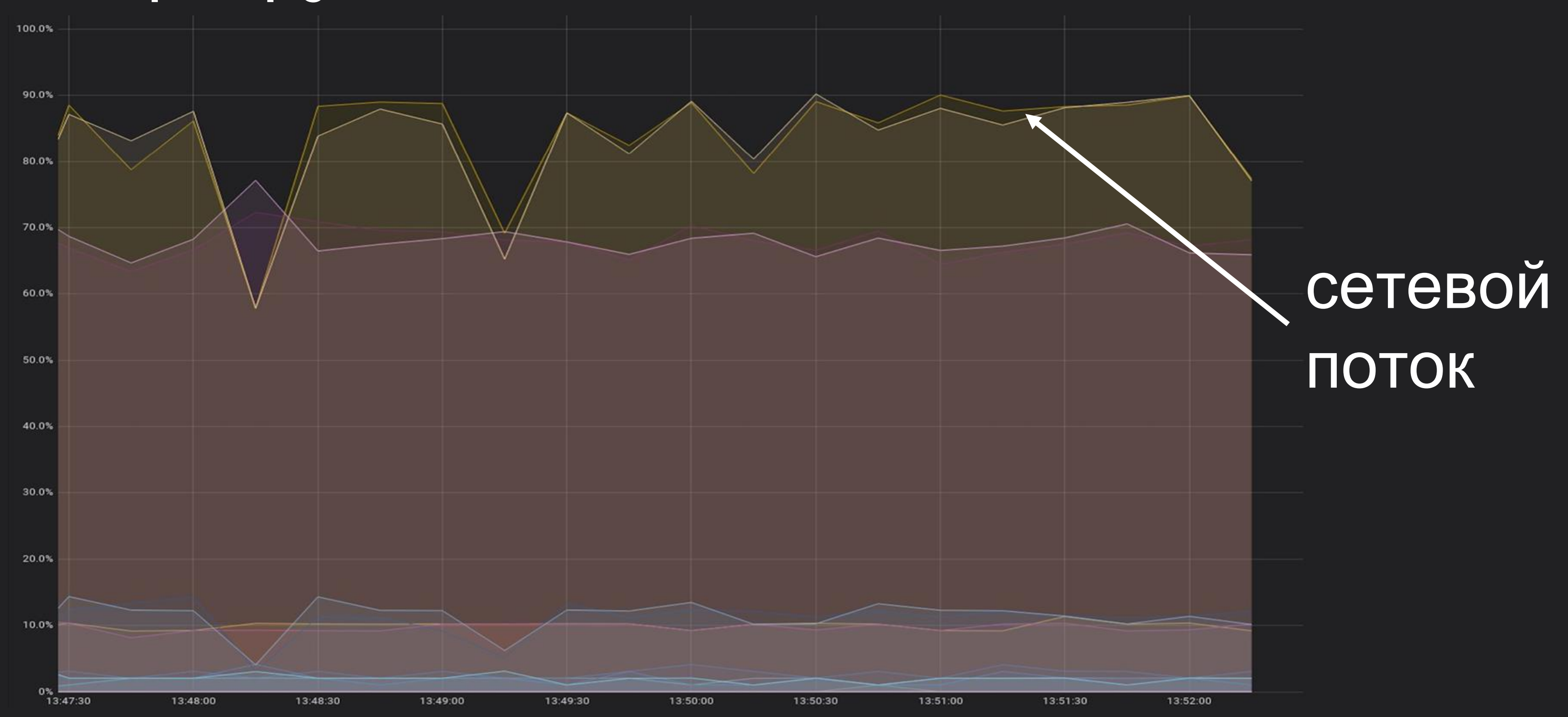


Рост отставания клиентов

резкий рост отставания клиентов



Перегрузка сетевого потока Tarantool



Что происходит?

Симптомы:

- Перегружен сетевой поток
- Выросли RPS клиента
- Клиентские запросы стали очень быстро обрабатываться



Поток данных

клиент читает данные

отставание клиента

новые данные из
торгового ядра



Поток данных

новые данные из
торгового ядра

клиент читает данные



Вывод:

Надо мониторить число пустых ответов



Начали мониторить количество отдаваемых пустых ответов

их оказалось более 80%

Они коррелируют с остальными симптомами



Вывод:

Надо минимизировать число пустых ответов



Первая версия реализации

Tarantool App

Торговое ядро

события об ордерах

ApplyEvent

вставка/обновление ордера

insert/update order

GetOrders

запрос ордеров после ***key***

select orders

Space (table)

get

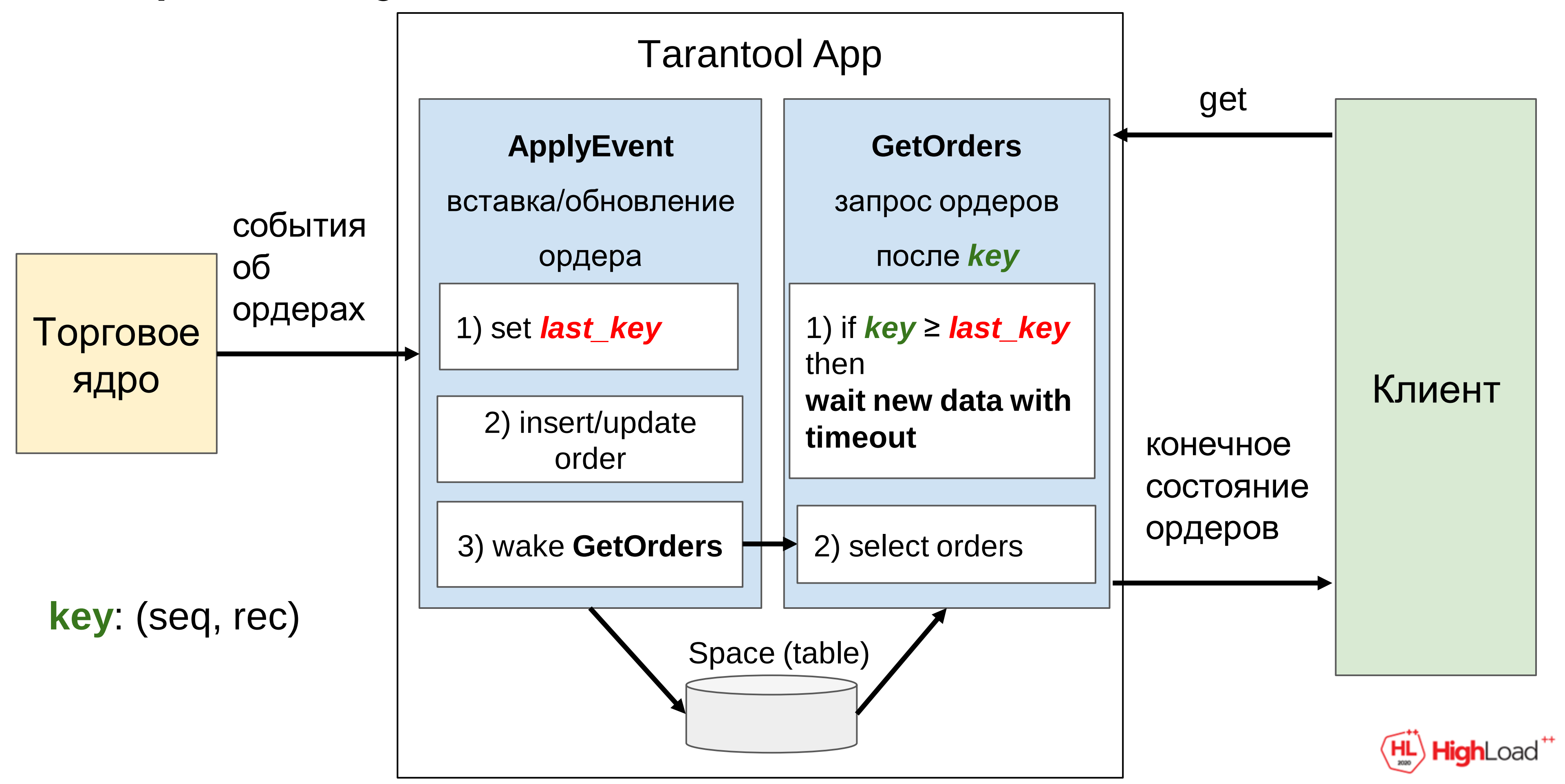
Клиент

конечное состояние ордеров

key: (seq, rec)



Версия с уведомлением о новых данных



- пустых ответов почти не стало
- на порядок снизилось количество пакетов в секунду
- снизилась нагрузка на сетевой поток на 15%



Вывод:

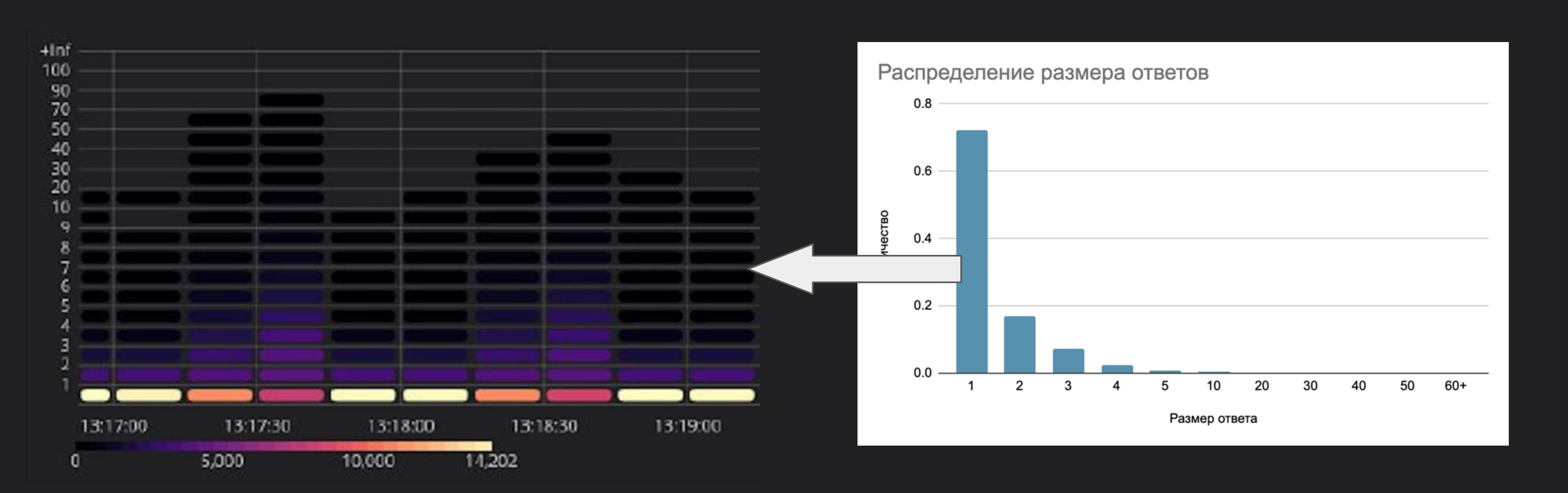
Число отдаваемых данных – важная метрика

Хотим ее контролировать

И измерять!



начинаем мерить размеры отдаваемых ответов



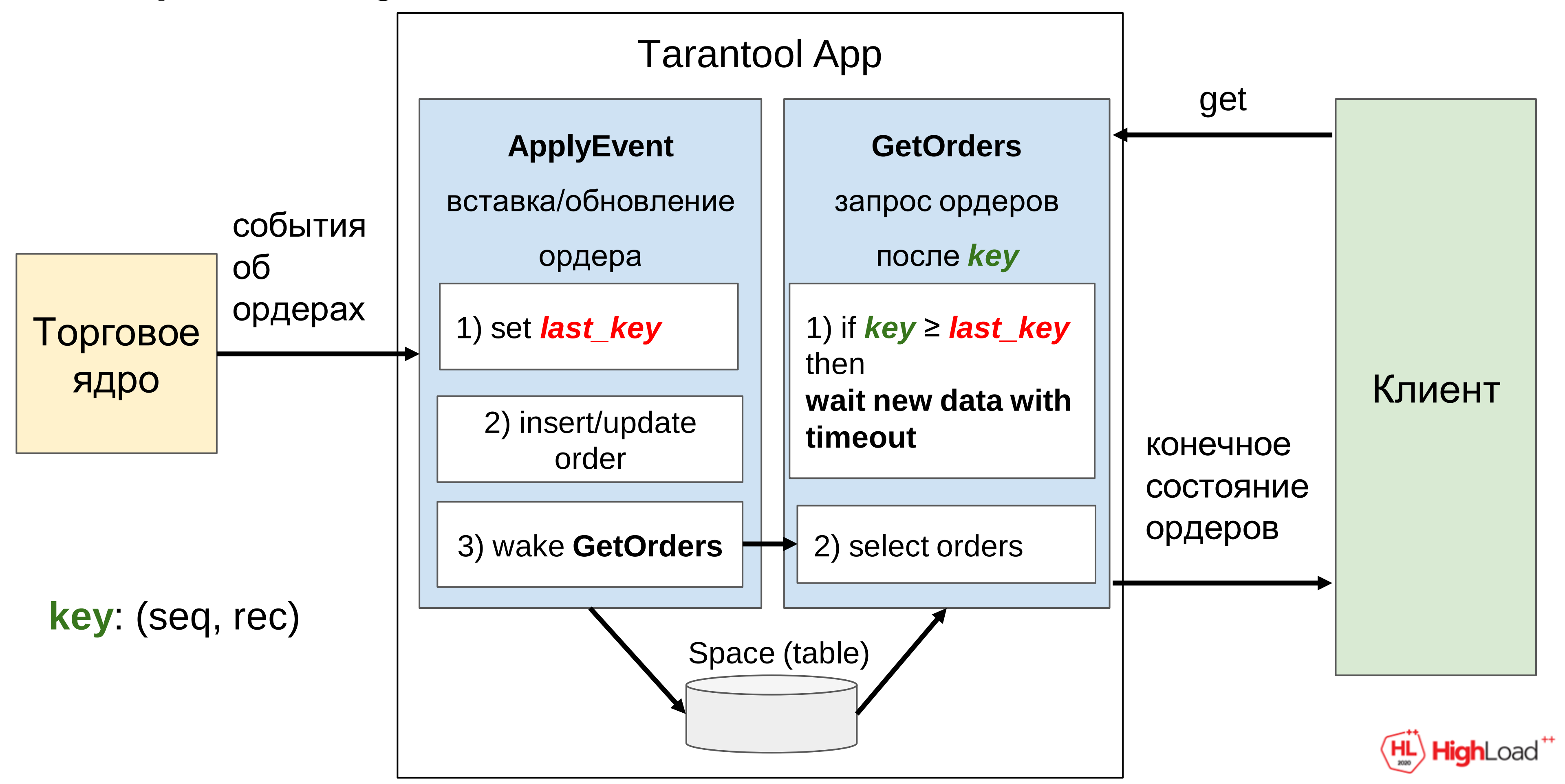
Распределение размера ответов

Хотим сдвинуть распределение вправо

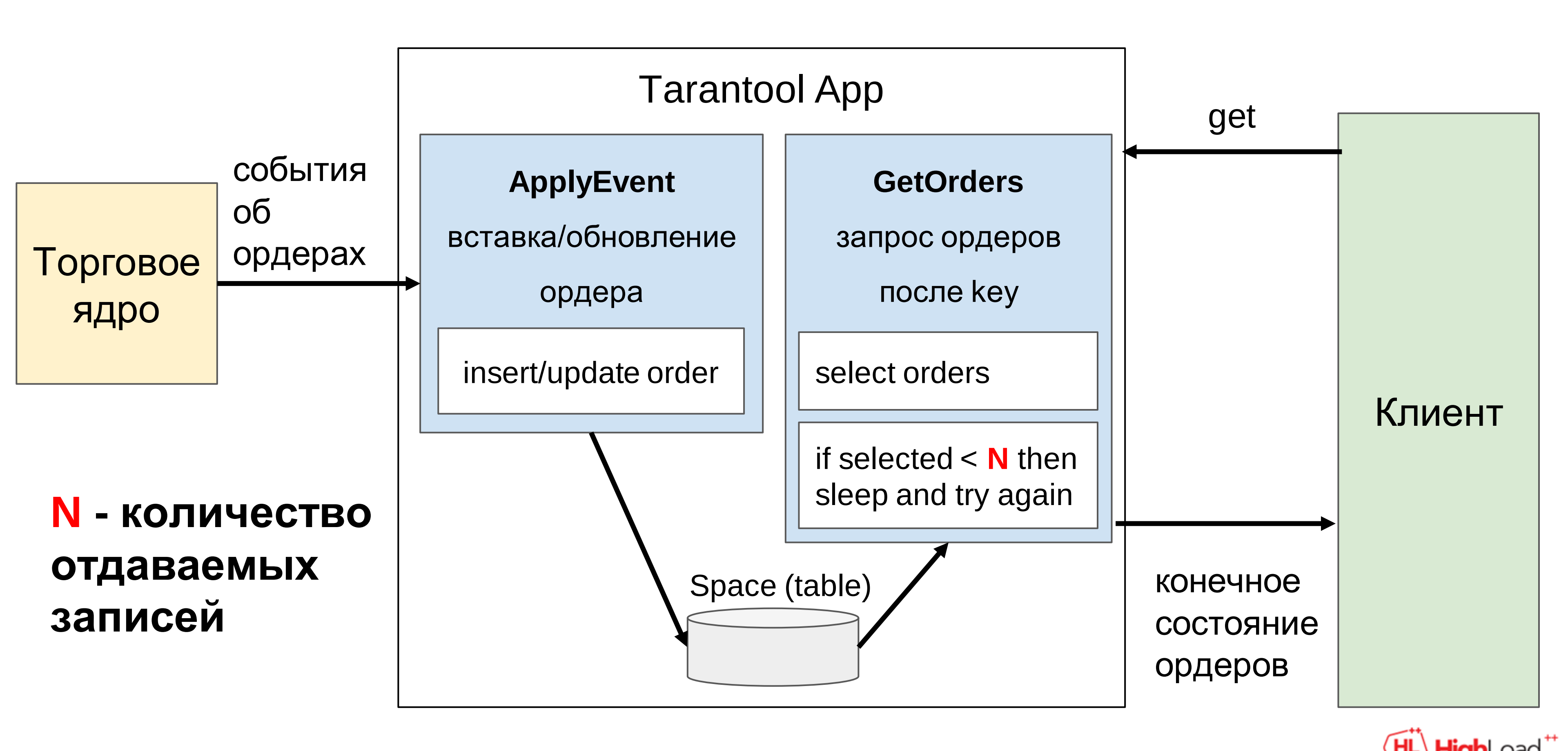
Чтобы клиенты забирали больше записей за 1 запрос



Версия с уведомлением о новых данных

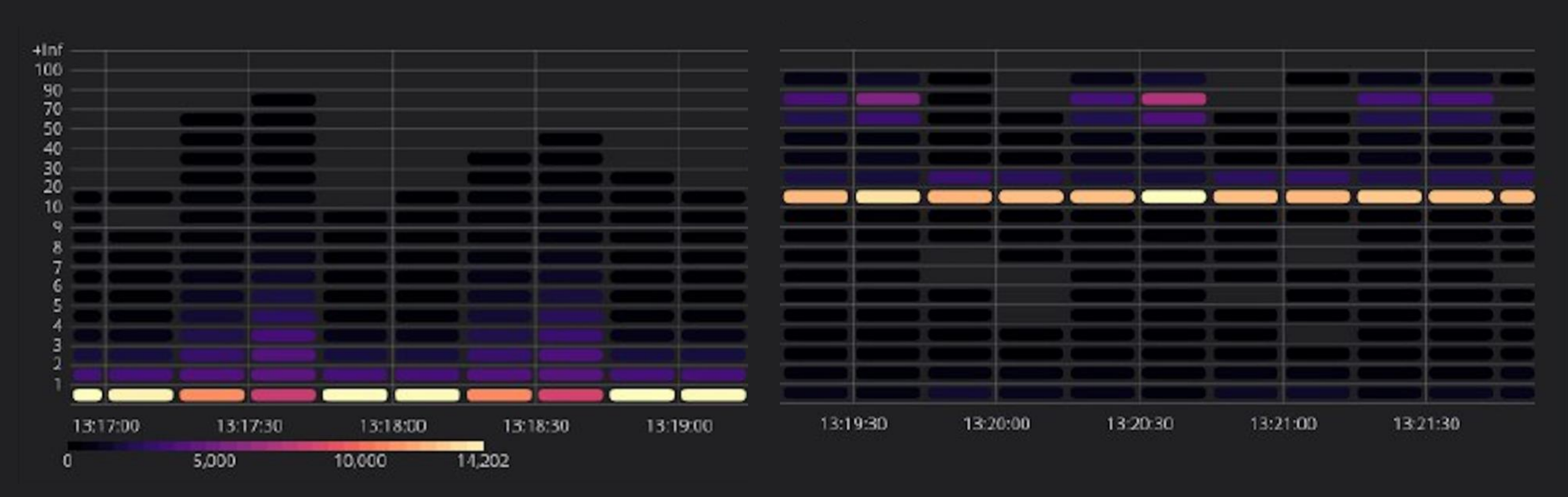


Версия со sleep



Было

Стало


+Inf
100
90
70
50
40
30
20
10
9
8
7
6
5
4
3
2
1
13:17:00
13:17:30
13:18:00
13:18:30
13:19:00
0
5,000
10,000
14,202
13:19:30
13:20:00
13:20:30
13:21:00
13:21:30

Максимальной поток чтения с записью

Запись, TPS	Кол-во клиентов без фильтров	CPU, %		Latency, ms			Размер пачки
		основной	сетевой	50	90	99	
170k	10	79	74	0.4	0.5	2.1	40
85k	15	56	63	0.3	0.4	1.0	20-70
85k	20	65	73	0.4	0.6	1.9	20-70



Сова снова ведет







3 : 1

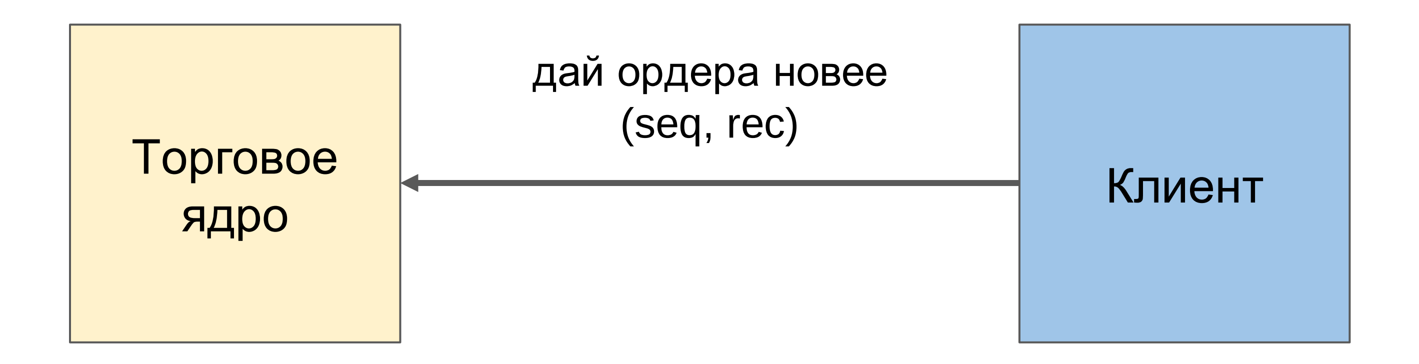


Сводим воедино требования к архитектуре

- Удельная производительность > 200 000 TPS на 1 узел
- Отставание от источника p99 < 5мс, а среднее — 1ms
- **Клиент не должен знать о разделении**
- **Должен гарантироваться порядок данных**
- Гарантия получения самых свежих данных без пропусков
- Одинаково хорошо отдавать как горячие (в пределах минуты), так и холодные (от минуты до дня)



Ключ запроса для монолита

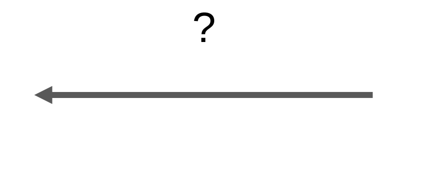




А если торговых ядер несколько?

Торговое ядро

Торговое ядро



Клиент

Теперь нет сквозных единых (rec, seq)



Что делать?

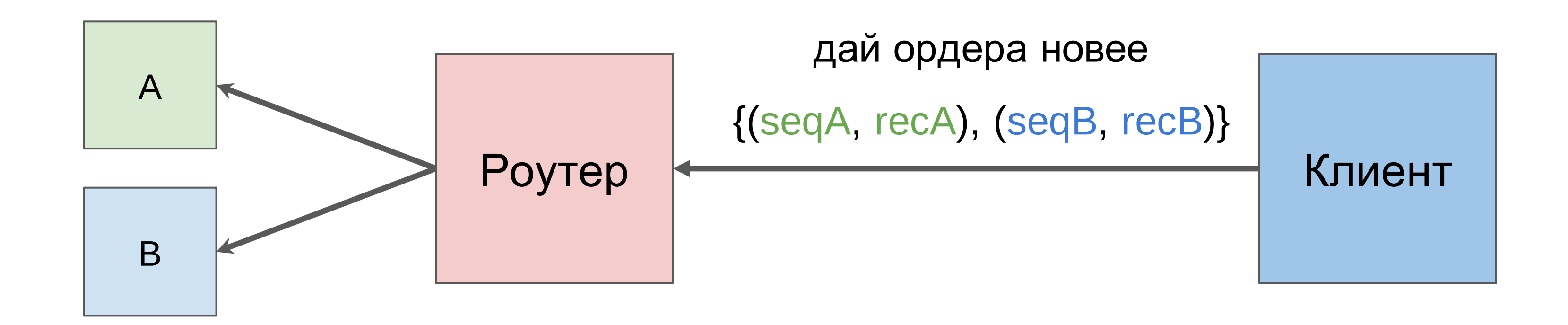
a) Ввести распределенные суррогатные req, sec
   + Не надо менять протокол
   - Либо это точка отказа
   - Либо нет согласованности
   - А если есть согласованность, то все медленно

b) Что-то придумать

Мы выбрали вариант b) – векторные часы



Распределенный ключ запроса



в качестве ключа запроса используются векторные часы:
{(seqA, recA), (seqB, recB)}

Минимальное изменение протокола, согласованнность



Источники

Кластер Тарантулов

Stateless-роутеры

Клиенты

A

B

C

Мастер

Мастер

Реплика

Клиент

Клиент

Клиент



Виды клиентов

Клиенты с фильтрацией и низкой задержкой

Клиенты без фильтрации



Источники

Кластер Тарантулов

Stateless-роутеры

Клиенты

A

B

C

Мастер

Мастер

Реплика

Клиент с фильтрацией и низкой задержкой

Клиент без фильтрации

Клиент без фильтрации



Источники

Кластер Тарантулов

Stateless-роутеры

Клиенты

A

B

C

Мастер

Мастер

Реплика

Клиент с фильтрацией и низкой задержкой

Клиент без фильтрации

Клиент без фильтрации



Источники

Кластер Тарантулов

Stateless-роутеры

Клиенты

A

B

C

Мастер

Мастер

Реплика

Клиент с фильтрацией и низкой задержкой

Клиент без фильтрации

Клиент без фильтрации



Источники

Кластер Тарантулов

Stateless-роутеры

Клиенты

A

B

C

Мастер

Мастер

Реплика **без индексов по фильтрам**

Клиент с фильтрацией и низкой задержкой

Клиент без фильтрации

Клиент без фильтрации



Тест с репликацией

Клиенты	Запись, TPS	CPU, %			Latency, ms		
		основной	**сетевой**	**репликатор**	**50**	**90**	**99**
Master (читают клиенты с фильтрацией и минимальной задержкой)							
5	60k	32	26	27	0.3	0.4	0.8
10	100k	66	60	46	0.3	0.7	4.6
Replica (читают клиенты без фильтрации)							
5	60k	36	15	-	0.4	0.7	1.7
10	100k	71	44	-	0.5	1.2	21.5



Сводим воедино требования к архитектуре

- [OK] Удельная производительность > 200 000 TPS на 1 узел
- **Отставание от источника p99 < 5мс, а среднее — 1ms**
- [OK] Клиент не должен знать о разделении
- [OK] Должен гарантироваться порядок данных
- [OK] Гарантия получения самых свежих данных без пропусков
- [OK] Одинаково хорошо отдавать как горячие (в пределах минуты), так и холодные (от минуты до дня)

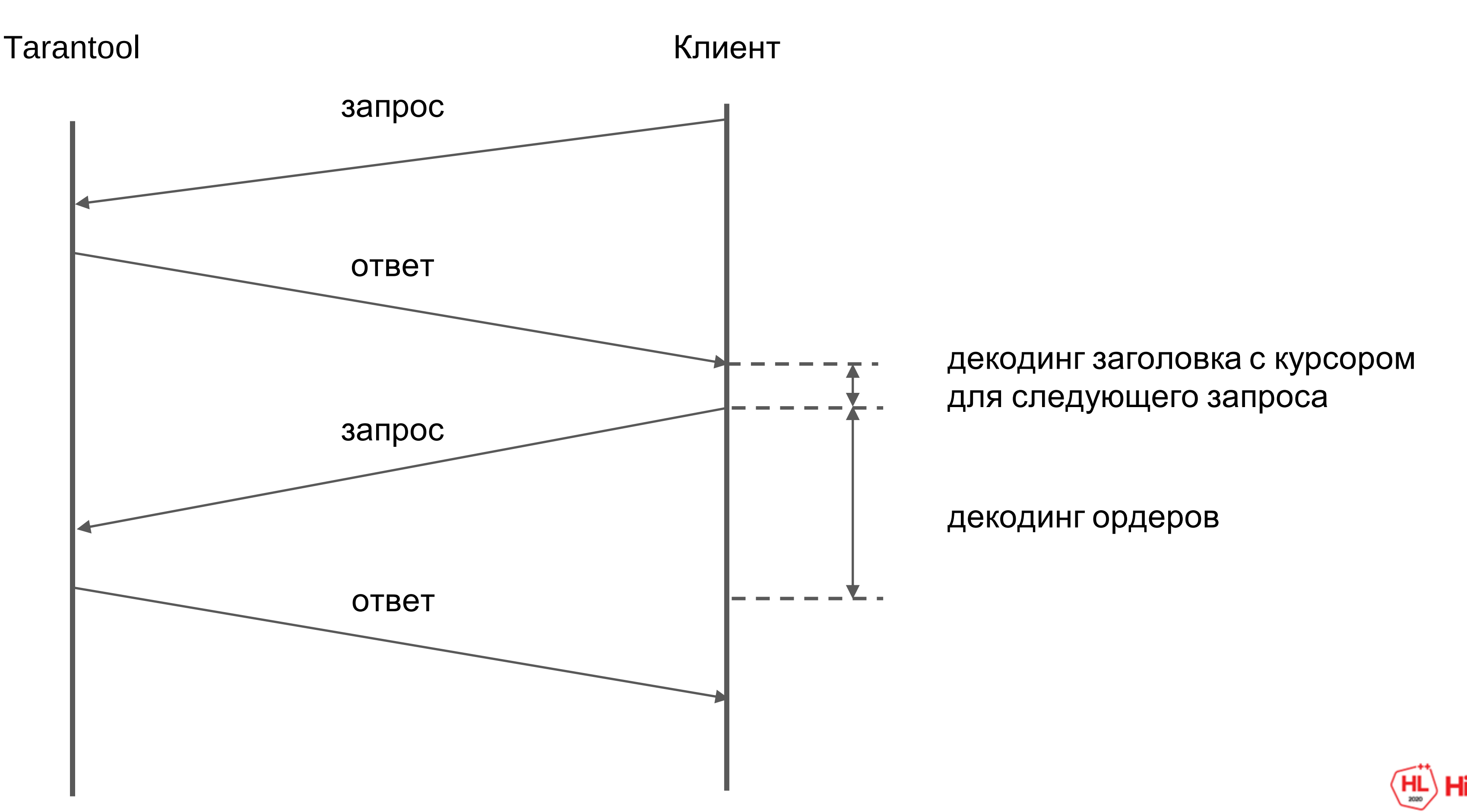


Тело ответа

request_received_ts request_decoded_ts storage_queried_ts ...	Трассировка (диагностика)
batch_last_seq batch_last_rec	курсор
oldest_order_time	время самой старой записи в пачке (диагностика)
record1 record2 record3 …	пачка ордеров



Процесс запроса



Эмуляторы

Эмулятор

Эмулятор

Эмулятор торгового ядра

события об ордерах

Кластер Tarantool

Tarantool

конечное состояние ордеров

Эмулятор клиента



Эмулятор торгового ядра (источника)

- генерация ордеров
- имитация всплесков
- Rate limiter
- подсчет RPS записи
- Prometheus exporter

Эмулятор клиента

- подсчет времени путешествия ордеров
- сбор трейсов из тарантула
- подсчет RPS чтения
- подсчет размера пачки
- Prometheus exporter

Приложение на Tarantool

- логика работы с данными
- сбор метрик CPU
- сбор метрик хранилища, в т.ч. insert/s, update/s, select/s, table size
- Prometheus exporter



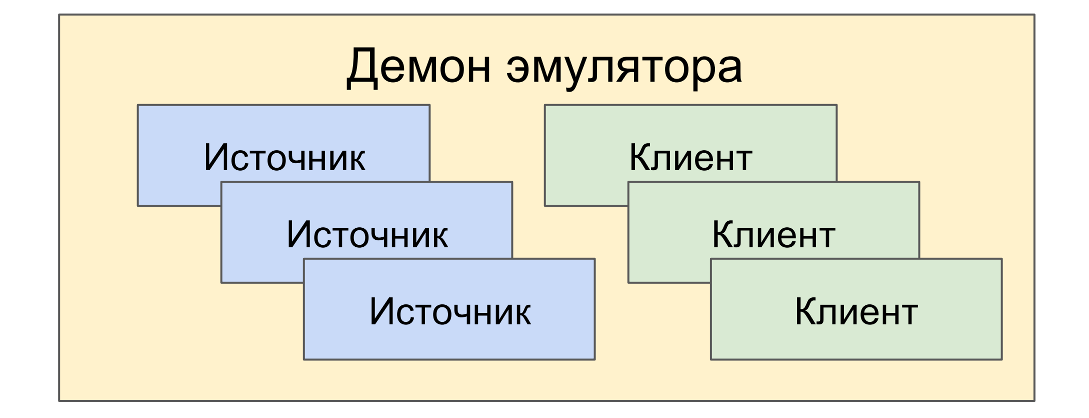
Эмуляторы

Было

Эмулятор торгового ядра

Эмулятор клиента

Стало





Автоматизация нагрузочного тестирования

Профиль тестирования и конфигурация кластера

Ansible

Демон эмулятора

Источник

Источник

Источник

Клиент

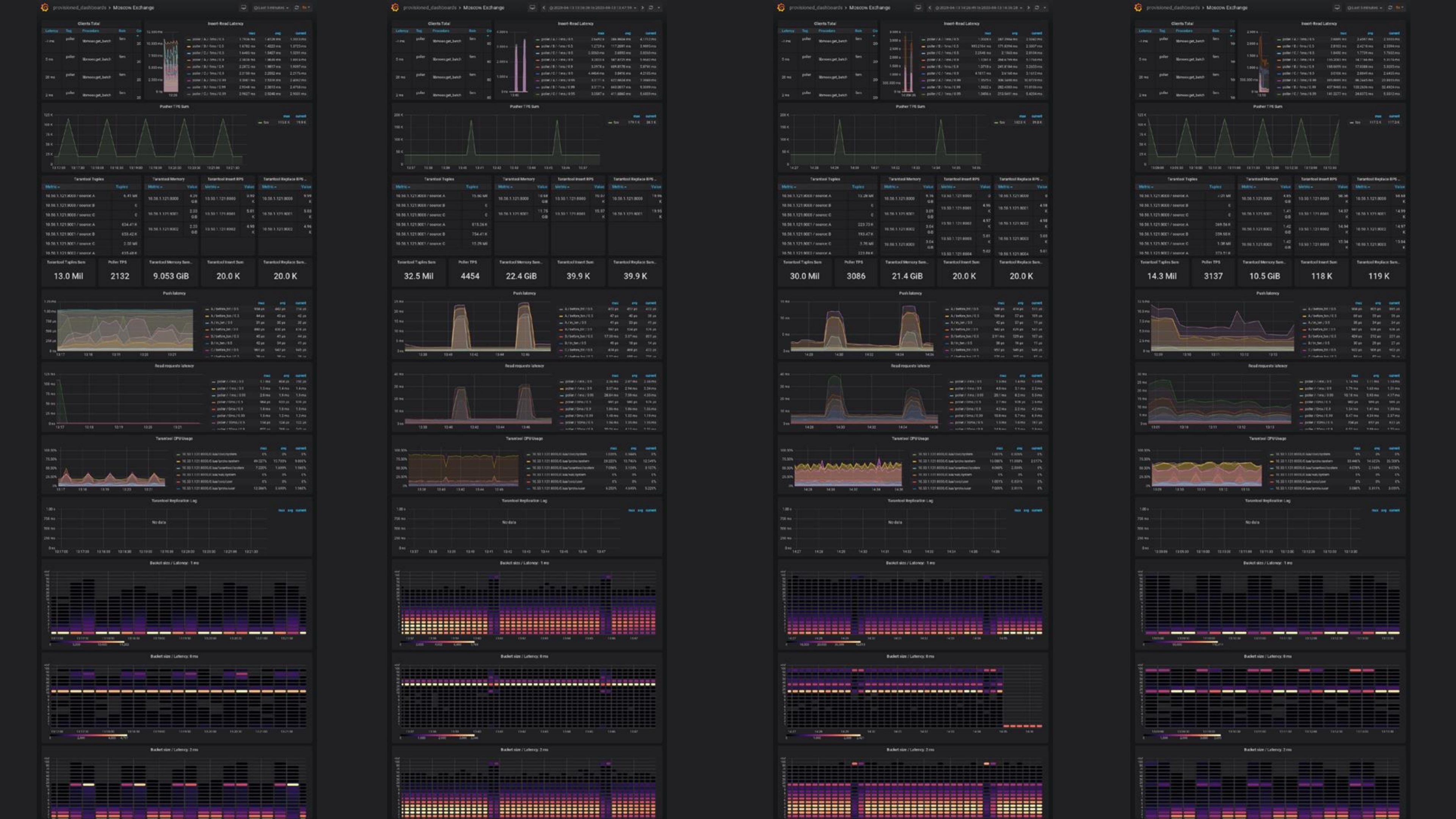
Клиент

Клиент

Кластер Тарантулов

Prometheus




provisioned_dashboards › Moscow Exchange
13.0 Mil
2132
9.053 GiB
20.0 K
20.0 K
provisioned_dashboards › Moscow Exchange
32.5 Mil
4454
22.4 GiB
39.9 K
39.9 K
provisioned_dashboards › Moscow Exchange
30.0 Mil
3086
21.4 GiB
20.0 K
20.0 K
provisioned_dashboards › Moscow Exchange
14.3 Mil
3137
10.5 GiB
118 K
119 K

Сводим воедино требования к архитектуре

- [OK] Удельная производительность > 200 000 TPS на 1 узел
- [OK] Отставание от источника p99 < 5мс, а среднее — 1ms
- [OK] Клиент не должен знать о разделении
- [OK] Должен гарантироваться порядок данных
- [OK] Гарантия получения самых свежих данных без пропусков
- [OK] Одинаково хорошо отдавать как горячие (в пределах минуты), так и холодные (от минуты до дня)



В итоге

- Пропилотирована архитектура распределенной системы доставки торговой информации с использованием Tarantool
- Пилот признан успешным
- В будущем планируется развитие



Сова победила







4 : 1



Выводы

- Есть проблема – надо научиться измерять
- Строить сложную систему итеративно от самого простого
- Измерять не только бизнес-метрики, но и технические метрики (ОС, железо)
- Изучать особенности алгоритмов обработки данных и структур данных инструментов
- Иногда полезно замедлять клиентов
- Батчить запросы, экономя CPU и сеть (в особенности пакеты)
- Использовать инструменты и технологии по назначению
- Автоматизировать тесты и забирать дамп из Prometheus ;)



Спасибо за внимание!



Вопросы?

Будем на связи:

nikolay.karlov@corp.mail.ru
o.utkin@corp.mail.ru

При подготовке этого прототипа ни одна сова не пострадала!

